You should work on the following assignments in fixed teams of two. Please note that *every* team member must be able to explain *all* solutions of the team of two. Please submit only one solution for each team of two. You can implement the assignments within the same programming project, i.e., you do not have to create a .c-file for every assignment. Please take care of the programming guideline (can be found in our moodle room) and handling of errors and special cases!

Deadline to upload your solution for at least assignments 1, 2, 3.1, 3.2, and 3.3:
Three days bevor the laboratory
Assignments 3.4 and 3.5 can be done during the laboratory.

If you have questions or if you need any help, use the forum in our moodle room und help each other.

This laboratory is about hashing. Two different strategies to handle collisions within the hash table are to be implemented:

- In Assignment 1: Address hashing
- In Assignment 3: Chained Hashing

For the solution of the assignments four files are available, which you should use:

- Lab3_students.cvs: A cvs-file containing students, where each student has a matriculation number as key and a name as value.
- Lab3.c: The file contains the main function. The user is asked which collision strategy he wants to use. Depending on this choice, the functions for address hashing or chained hashing have to be applied for inserting, searching, etc.
- Lab3_HashAHLib.h: This is the header file with the macros, data type, and function prototypes needed for address hashing.
- Lab3_HashAHLib.c: In this file you should implement the functions needed for address hashing.

Assignment 1: Hashing with "address hashing" as collision strategy

1.1 Implement the function putAH

The function creates a new entry in the hash table or if the key already exists overwrites the value with the passed value. The function receives the hash table and the key-value-pair to be inserted. If a collision is detected it uses the address hashing as collision strategy. I.e., the table is probed starting from the occupied slot in a specified manner until an open slot is located or the entire table is probed (hash table is already full), here linear probing should be used. Linear probing means the function searches for the next free entry in the hash table starting from the occupied slot. The function returns the number of collisions. For information see the lecture slides and Figure 1.

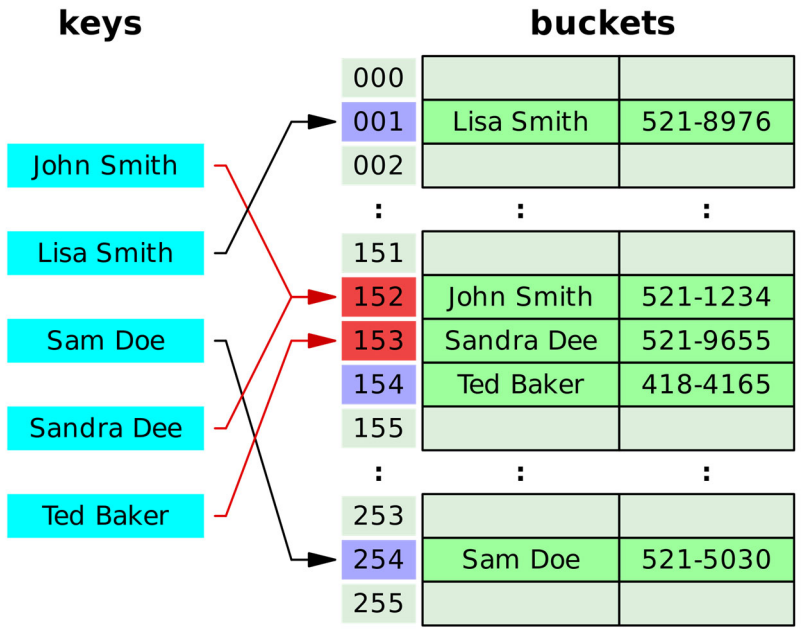

Figure 1: Address Hashing (source: https://en.wikipedia.org/wiki/Hash_function)

1.2 Implement function `getAH`

The function searches for the entry in the hash table having the passed key.
It returns the value of the entry or `NULL` if the key does not exist in the hash table.

1.3 Implement function `deleteAH`

The function searches for the entry in the hash table having the passed key.
If it finds an entry the function deletes it from the hash table.

1.4 Implement function `printHashTableAH`

The function prints the hash table. If a field in the array is empty, it prints "*empty*".

Assignment 2 Function `hashing`

Play with the hash function. Try at least three other ways to calculate the hash value than the simple division hashing that is used in the given function `hashing`. How do the other calculations behave? E.g., can the number of collisions be reduced?

Assignment 3: Hashing with "chained hashing" as collision strategy

Create a new library for chained hashing, consisting of the files HashCHLib.h and HashCHLib.c.

3.1. Data structure `sElementCH`

Define a data structure `sElementCH` that can be used for chained hashing. Please keep in mind how the chained hashing works (see Assignment 3.3).

3.2 Create hash table

In `main` function the hash table and its initialization have to be implemented in switch case 2, i.e., if the user chooses the chained hashing as collision strategy.
If you like you can also implement function `readCSVCH` by copying and slightly adapting function `readCSVAH` provided in file HashAHLib.c. Then you can reuse the csv-file students.csv and read in the students in your hash table.

3.3 Implement the function `putCH`

The function creates a new entry in the hash table or if the key already exists overwrites the value with the passed value. The function receives the hash table and the key-value-pair to be inserted. If a collision is detected it uses the chained hashing as collision strategy. I.e., each slot is the head of a linked list (*chain*), and items that collide at the slot are added to the list. If there is a collision the function adds the new entry as first entry (head) of the list. For information see the lecture slides and Figure 2.

Note: in Figure 2, in case of a collision the function appends the new entry to the end of the list. As this is more difficult to implement you can also add the new entry as head of the chain.

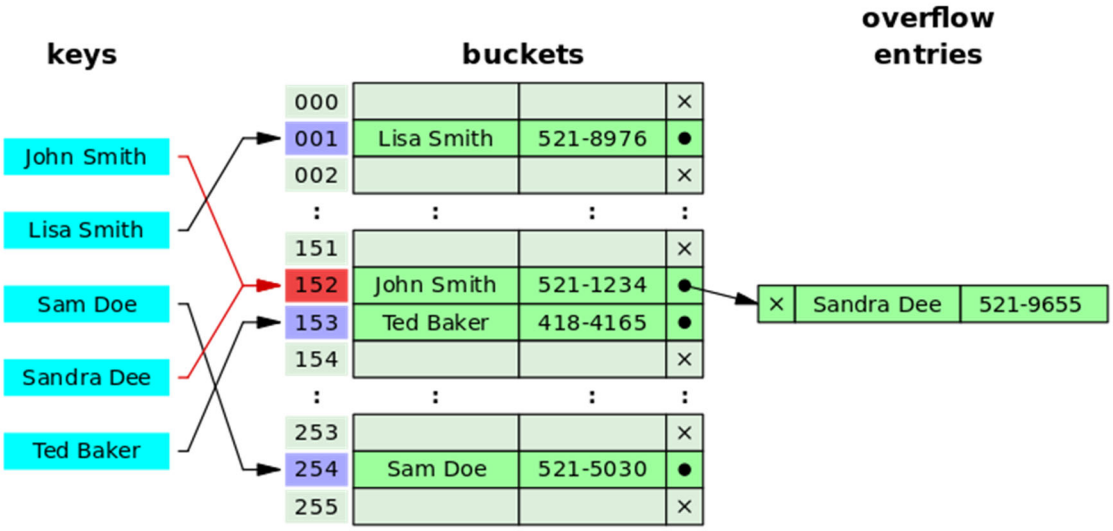

Figure 2: Chained Hashing (source: https://en.wikipedia.org/wiki/Hash_function)

3.4 Implement function `getCH`

The function searches for the entry in the hash table having the passed key.
It returns the value of the entry or `NULL` if the key does not exist in the hash table.

3.5 Implement function `printHashTableCH`

The function prints the hash table. If a field in the array is empty, it prints "*empty*".